****

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«МИРЭА – Российский технологический университет»**

# РТУ МИРЭА

Отчет по выполнению практического задания №8.1

**Тема: «ОТДЕЛЬНЫЕ ВОПРОСЫ АЛГОРИТМИЗАЦИИ»**

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент Кузнецов Л. А.

группа ИКБО-20-23

**Москва 2024**

**СОДЕРЖАНИЕ**

[ЧАСТЬ 8.1………………………………………………………………………..3](#часть81)

[Задание 1…………………………………………………………………………3](#задание1)

[Вариант…………………………………………………………………………...3](#вариант1)

[Решение…………………………………………………………………………..3](#решение1)

[Задание 2…………………………………………………………………………6](#задание2)

[Решение…………………………………………………………………………..6](#решение2)

[Тесты…………………………………………………………………………….10](#тесты)

[ВЫВОД………………………………………………………………………….13](#вывод)

[СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ…………………………14](#список)

**ЧАСТЬ 8.1**

**Задание 1**

1. Выполнить сжатие элементов по методу Лемпеля-Зива LZ77 используя двухсимвольный алфавит.
2. Закодировать фразу, используя код LZ78.
3. Описать процесс сжатия/кодировки

**Вариант**

18 вариант.

Код для сжатия элементов по методу Лемпеля-Зива LZ77:

1110100110111001101.

Фраза для кодировки при помощи кода LZ78: sionsinossionsinos.

Фраза для кодировки методами Шеннона-Фано: “Дрынцы-брынцыбубенцы, Раззвонилисьудальцы, Диги-диги-диги-дон, Выхо-ди-скорее-вон!”.

**Решение**

Начнём с сжатия элементов по методу Лемпеля-Зива LZ77.

Отобразим при помощи таблицы 1 ход присвоения кода для конкретной последовательности.

Таблица 1 – Ход присвоения последовательностям кода

|  |  |  |
| --- | --- | --- |
| Содержимое окна (сжимаемый текст) | Содержимое упреждающего буфера | Код назначенный последовательности |
|  | 1110100110111001101 |  |
| **1** | 110100110111001101 | **1** |
| **11** | 0100110111001101 | **10** |
| 01 | 00110111001101 | **11** |
| 00 | 110111001101 | **100** |
| 110 | 111001101 | **101** |
| 111 | 001101 | **110** |
| 001 | 101 | **111** |
| 101 |  | **1000** |

Теперь же перейдём к составлению конечной последовательности как символов, так и кодов.

В первые две очереди последовательности кодов запишем сначала 1, а после 11, так как обе эти комбинации уникальны

Послед-ость: 1.11.

Теперь присвоим последовательности комбинацию 01 – она уникальна, а её код длины 2, поэтому записываем её без изменений.

Послед-ость: 1.11.01.

Далее идёт комбинация 00 с кодом 100. Комбинация уникальна, но её код длины 3, поэтому увеличиваем длину 00 до 000 и записываем полученную комбинацию в последовательность.

Послед-ость: 1.11.01.000.

После переходим к комбинации 110 – она не уникальна, поэтому присваиваем ей код 10 от комбинации 11 и дописываем 0, чтобы длина полученной комбинации была 3

Послед-ость: 1.11.01.000.100.

Далее следует 111, которой присваивается код 10 от 11 и дописывается 1 в конце.

Послед-ость: 1.11.01.000.100.101.

Рассмотрим комбинацию 001. Сначала присвоим ей код 100 от 00, а после допишем 1.

Послед-ость: 1.11.01.000.100.101.1001.

И последний элемент – 101. Он уникален, но из-за того, что длина предыдущей последовательности 4, то и эта последовательность должна иметь не меньшую длину. Тогда в итоге получим 0101.

Послед-ость: 1.11.01.000.100.101.1001.0101.

В конце всех преобразований мы получили следующие последовательность и её код.

**Послед-ость: 1.11.01.000.100.101.1001.0101**

**Её код: 1.10.11.100.101.110.111.1000**

Теперь же перейдём к кодированию фразы при помощи кода LZ78.

Фраза: sionsinossionsinos

Отобразим результаты кодировки при помощи таблицы 2.

Таблица 2 – Процесс кодировки фразы методом LZ78

|  |  |  |
| --- | --- | --- |
| Словарь | Считываемое содержимое | Код |
|  | s | <0, s> |
| s = 1 | i | <0, i> |
| s = 1, i = 2 | o | <0, o> |
| s = 1, i = 2, o = 3 | n | <0, n> |
| s = 1, i = 2, o = 3, n = 4 | si | <1, i> |
| s = 1, i = 2, o = 3, n = 4 si= 5 | no | <4, o> |
| s = 1, i = 2, o = 3, n = 4 si= 5, no = 6 | ss | <1, s> |
| s = 1, i = 2, o = 3, n = 4 si= 5, no = 6, ss = 7 | io | <2, o> |
| s = 1, i = 2, o = 3, n = 4 si= 5, no = 6, ss = 7, io= 8 | ns | <4, s> |
| s = 1, i = 2, o = 3, n = 4 si= 5, no = 6, ss = 7, io= 8 ns = 9 | in | <2, n> |
| s = 1, i = 2, o = 3, n = 4 si= 5, no = 6, ss = 7, io= 8 ns = 9, in = 10 | os | <3, s> |
| s = 1, i = 2, o = 3, n = 4 si= 5, no = 6, ss = 7, io= 8 ns = 9, in = 10 |  | <0, EOF> |

Как видно из таблицы 2, мы постепенно проходились по фразе, убирая из неё уже увиденные комбинации, и добавляли в словарь неповторяющиеся.

**Итоговая послед-ость: <0, s>, <0, i>, <0, o>, <0, n>, <1, i>, <4, o>, <1, s>, <2, o>, <4, s>, <2, n>, <3, s>, <0, EOF>.**

**Задание 2**

1. Разработать программы сжатия и восстановления текста методами Хаффмана и Шеннона-Фано.
2. Описать разработанные методы и провести тестирование.

**Решение**

Для реализации метода Шеннона-Фано сначала посчитаем кол-во каждого из символов, после отсортируем полученные значения по невозрастанию, затем делим полученные значения по частям и строим дерево с символами и выводим его на экран.

После выяснения стратегии решения задачи был написан код.

В первую очередь была разработана логика структуры dict, которая хранит в себе значения кол-ва повторений конкретного символа и выполняет работу словаря (рис. 1-3).

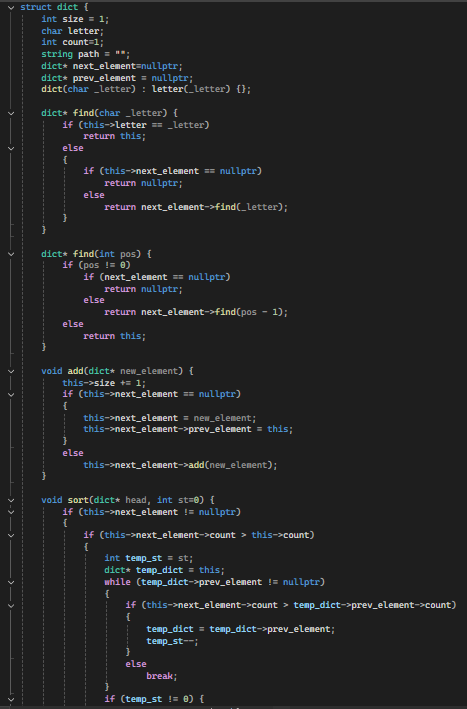


Рисунок 1 – 1-ая часть словаря dict

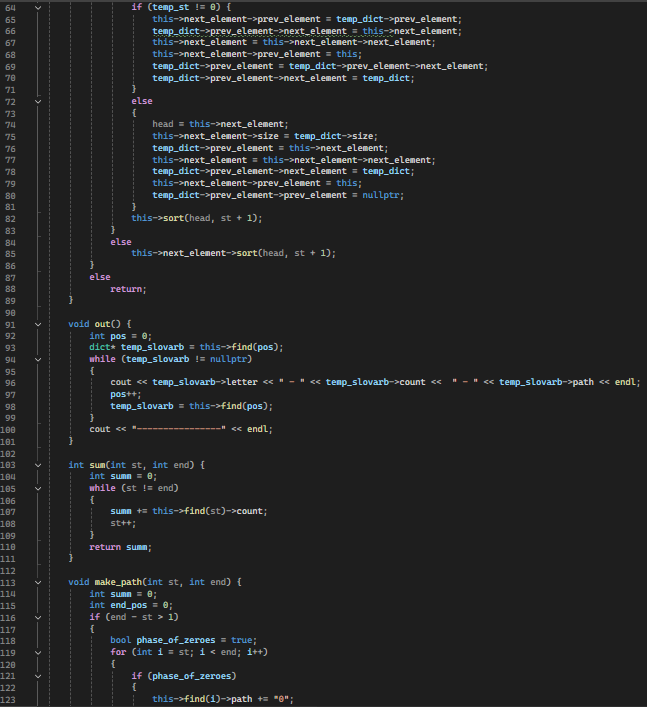


Рисунок 2 – 2-ая часть словаря dict

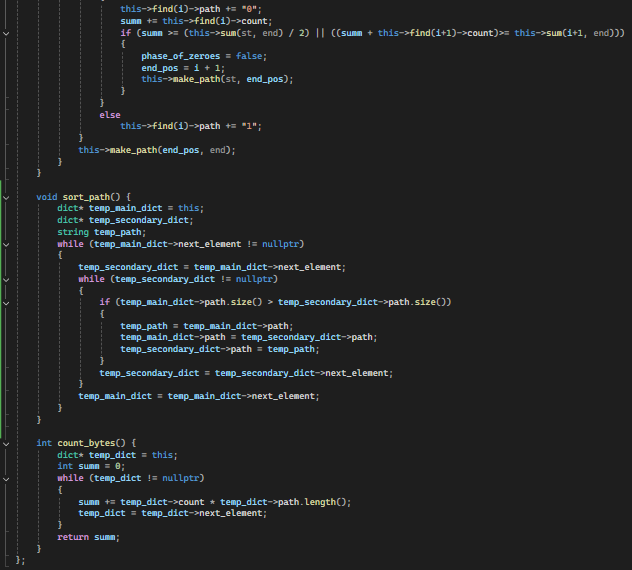


Рисунок 3 – 3-я часть словаря dict

Данный словарь реализует все необходимые взаимодействия с символами: создаёт путь для дерева, хранит значения повторений, а также ссылку на следующий элемент, осуществляет поиск элементов по индексу и символу, совершает вывод данных на экран, а также находит сумму бит всех данных.

Как оказывается, прописка логики словаря – это самая сложная и интересная часть данной работы.

Далее последовала реализация логики листьев будущего дерева, а именно структуры leaf (рис. 4).

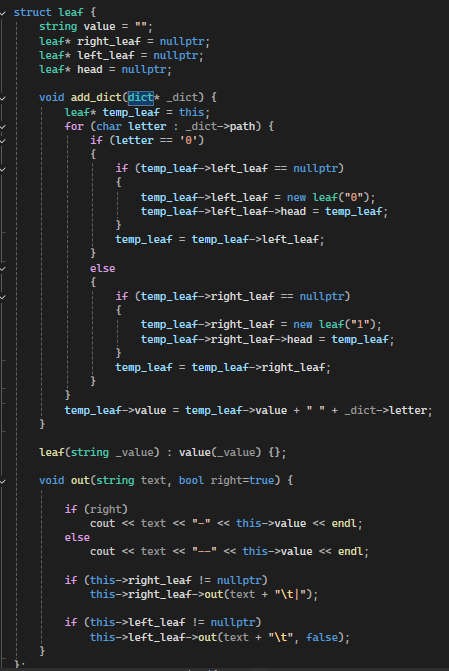


Рисунок 4 – Реализация логики структуры leaf

Данные листья по своей сути и являются самим деревом. Что имеется под этим ввиду? Имеется ввиду то, что дерево заранее не определено и оно формируется в зависимости от путей элементов структуры dict, которая впоследствии задаёт само дерево через структуру leaf.

И последней частью реализации работы программы в соответствии с заданными алгоритмами является реализация работы функции main (рис. 5, 6).

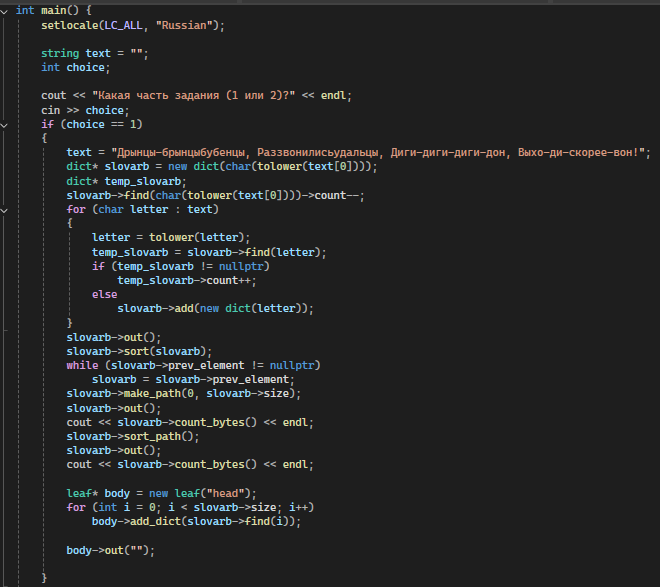


Рисунок 5 – 1-ая часть реализации алгоритма работы функции main

В первой части представлена работы программы с методом Шеннона-Фано, а во втором – методом Хаффмана.

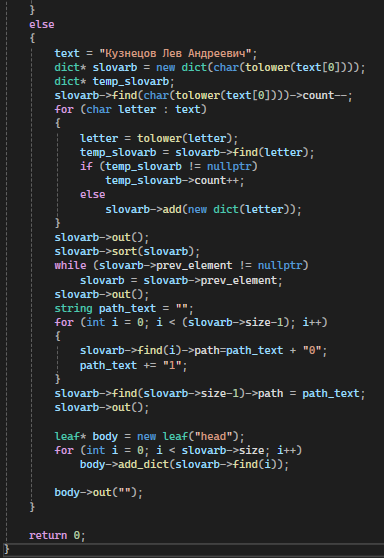


Рисунок 6 – 2-ая часть реализации алгоритма работы функции main

Как можно заметить программа работает с текстом и кодирует его в соответствии с заданным текстом, что и требовалось выполнить.

Теперь настало время провести необходимые тесты.

**Тесты**

Для начала проведём тесты с алгоритмом Шеннона-Фано.

Фраза была описана выше.

Начнём тест с того, что подсчитаем все значения повторений элемнтов, выведем сжатый и первоначальный вариант кол-ва битов. (рис. 7)

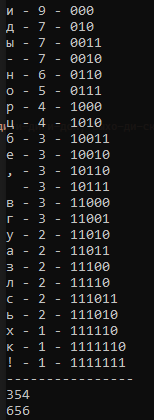


Рисунок 7 – 1-ая часть результатов тестов метода Шеннона-Фано

Как видно из рисунка 7 текст получилось сжать практически в два раза.

Далее выведем на экран дерево с отображёнными символами на нём (рис. 8).

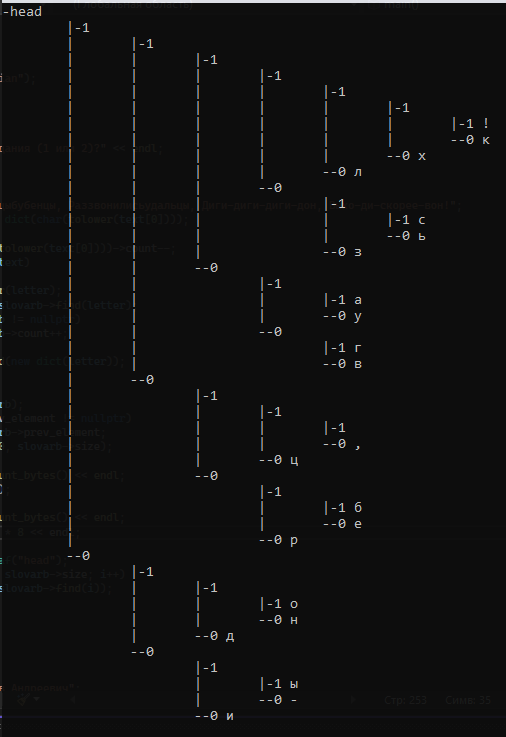


Рисунок 8 – 2-ая часть результатов тестов метода Шеннона-Фано

На рисунке 8 видно, что корень дерева отмечен как head, чтобы было проще сориентироваться, далее к каждому листу присоединено его значение пути (то есть 1 или 0), а также символ, которому соответствует этот путь.

Составим таблицу 3 вероятностей для данного случая.

Таблица 3 – Таблица вероятности появления символа в тексте

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Алфавит | и | д | ы | - | н | о | р | ц |
| Кол. вх. | 9 | 7 | 7 | 7 | 6 | 5 | 4 | 4 |
| Вероятн. | 0,11 | 0,08 | 0,08 | 0,08 | 0,07 | 0,06 | 0,05 | 0,05 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Алфавит | б | е | , | « » | в | г | у | а |
| Кол. вх. | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 |
| Вероятн. | 0,04 | 0,04 | 0,04 | 0,04 | 0,04 | 0,04 | 0,02 | 0,02 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Алфавит | з | л | с | ь | х | к | ! |
| Кол. вх. | 2 | 2 | 2 | 2 | 1 | 1 | 1 |
| Вероятн. | 0,02 | 0,02 | 0,02 | 0,02 | 0,01 | 0,01 | 0,01 |

Теперь проведём тесты с методом Хаффмана (рис. 9).

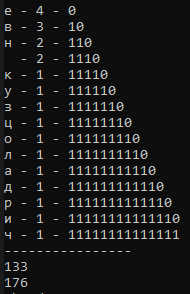


Рисунок 9 – 1-ая часть результатов тестов метода Хаффмана

Как видно из тестов, текст не сильно-то и сжался, однако алгоритм реализации подобного метода является чрезвычайно простым в исполнении.

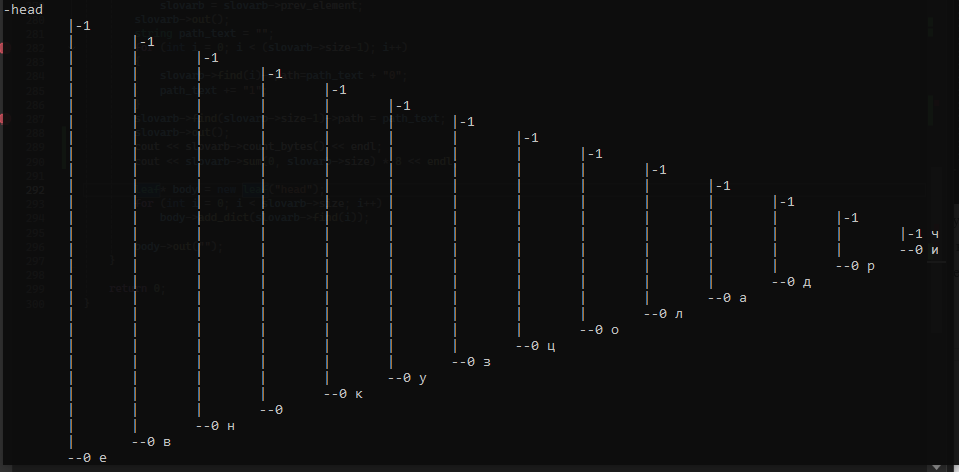


Рисунок 9 – 2-ая часть результатов тестов метода Хаффмана

Можно заметить, что рисунок 9 отображает закономерную структуру формирования путей для каждого из символов, коей и является данная структура.

После тестов была написана таблица 4 – таблица вероятностей.

Таблица 4 – Таблица вероятности появления символа в тексте

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Алфавит | е | в | н | « » | к | у | з | ц |
| Кол. вх. | 4 | 3 | 2 | 2 | 1 | 1 | 1 | 1 |
| Вероятн. | 0,27 | 0,20 | 0,13 | 0,13 | 0,07 | 0,07 | 0,07 | 0,07 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Алфавит | о | л | а | д | р | и | ч |
| Кол. вх. | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Вероятн. | 0,07 | 0,07 | 0,07 | 0,07 | 0,07 | 0,07 | 0,07 |

**ВЫВОД**

Изучили различные методы сжатия текста, а также применили полученные знания на практике, приобретя новые навыки и проведя тесты в соответствии с вариантом задания.

**СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ**

1. Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.

2. Документация по языку С++ [Электронный ресурс]. URL: https://docs.microsoft.com/ruru/cpp/cpp/ (дата обращения 01.09.2021).

3. Курс: Структуры и алгоритмы обработки данных. Часть 2 [Электронный ресурс]. URL: https://online-edu.mirea.ru/course/view.php?id=4020 (дата обращения 01.09.2021).